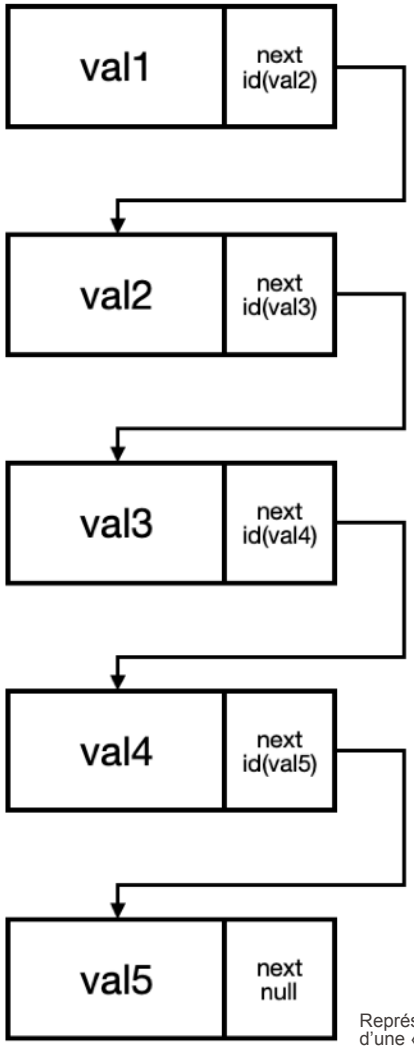


Représentation simplifiée d'une «liste chaînée»

Information, Calcul et Communication

Partie Programmation

Cours 4: Fonctions – Listes

13.10.2023

Patrick Wang

Au sommaire cette semaine

1. Quelques compléments sur les fonctions
2. Nouvelle structure de données : les listes

Au sommaire cette semaine

1. Quelques compléments sur les fonctions
2. Nouvelle structure de données : les listes

1. Quelques compléments sur les fonctions

Retour sur un exercice de la semaine dernière

- L'exercice 6 de la semaine passée a introduit les default argument values et les keyword arguments.
- Observez le programme suivant, quel sera le résultat de son exécution ?

```
def substract(x: int, y: int) -> int:
    return x - y

x: int = 1
y: int = -1

print(substract(x, y))
print(substract(y, x))
print(substract(y=y, x=x))
print(substract(y=x, x=y))
```

1. Quelques compléments sur les fonctions

Ordre des paramètres

- Dans la définition et dans l'appel d'une fonction, l'ordre des paramètres est important !

```
def substract(x: int, y: int) -> int:
    return x - y

x: int = 1
y: int = -1

print(substract(x, y))
print(substract(y, x))
print(substract(y=y, x=x))
print(substract(y=x, x=y))
```

1. Quelques compléments sur les fonctions

Ordre des paramètres

- Dans la définition et dans l'appel d'une fonction, l'ordre des paramètres est important !

```
def substract(x: int, y: int) -> int:
    return x - y

x: int = 1
y: int = -1

print(substract(x, y))
print(substract(y, x))
print(substract(y=y, x=x))
print(substract(y=x, x=y))
```

1. Quelques compléments sur les fonctions

Ordre des paramètres

- Dans la définition et dans l'appel d'une fonction, l'ordre des paramètres est important !

```
def substract(x: int, y: int) -> int:
    return x - y

x: int = 1
y: int = -1

print(substract(x, y))
print(substract(-1, 1))
print(substract(y=y, x=x))
print(substract(y=x, x=y))
```

1. Quelques compléments sur les fonctions

Ordre des paramètres

- Dans la définition et dans l'appel d'une fonction, l'ordre des paramètres est important !

```
def substract(x: int, y: int) -> int:
    return (-1) - (1)

x: int = 1
y: int = -1

print(substract(x, y))
print(substract(-1, 1))
print(substract(y=y, x=x))
print(substract(y=x, x=y))
```

1. Quelques compléments sur les fonctions

Ordre des paramètres

- Dans la définition et dans l'appel d'une fonction, l'ordre des paramètres est important ! (sauf si on utilise les keyword arguments)

```
def substract(x: int, y: int) -> int:
    return x - y

x: int = 1
y: int = -1

print(substract(x, y))
print(substract(y, x))
print(substract(y=y, x=x))
print(substract(y=x, x=y))
```

1. Quelques compléments sur les fonctions

Noms des paramètres et des arguments

- Les noms des paramètres n'engagent à rien !
- Les x, y des paramètres n'ont rien à voir avec les variables x, y.

```
def substract(x: int, y: int) -> int:
    return x - y

x: int = 1
y: int = -1

print(substract(x, y))
print(substract(y, x))
print(substract(y=y, x=x))
print(substract(y=x, x=y))
```

1. Quelques compléments sur les fonctions

Noms des paramètres et des arguments

- Les noms des paramètres n'engagent à rien !
- Les x, y des paramètres n'ont rien à voir avec les variables x, y.

```
def substract(x: int, y: int) -> int:
    return x - y

a: int = 1
b: int = -1

print(substract(a, b))
print(substract(b, a))
print(substract(y=b, x=a))
print(substract(y=a, x=b))
```

Au sommaire cette semaine

1. Quelques compléments sur les fonctions
2. Nouvelle structure de données : les listes

2. Nouvelle structure de données : les listes

Problème initial

- Les variables utilisées jusqu'à maintenant ne contiennent qu'une seule valeur.
- Existe-t-il une structure de données permettant de stocker et manipuler plusieurs valeurs ?
- Pour rappel : « une variable est un emplacement mémoire identifiable grâce à un nom ».

2. Nouvelle structure de données : les listes

Les listes – Introduction et syntaxe

- Une liste est une structure de données contenant un nombre indéfini de valeurs (a priori du même type).
- Utilisation des [ ] pour définir une liste, avec des virgules pour séparer les éléments.
- Ces éléments peuvent être récupérés grâce à leur indice dans la liste.

```
from typing import List

# Déclaration d'une liste, avec des valeurs initiales
numbers: List[float] = [10.5, 32.0, -2, 23/2]

# Déclaration d'une liste vide, puis construction au fur et à mesure
numbers: List[float] = []
numbers.append(10.5)
numbers.append(32.0)
numbers.extend([-2, 32/2])
```

2. Nouvelle structure de données : les listes

Arrêt sur le vocabulaire

- Une méthode s'appelle sur une variable pour effectuer une séquence d'instructions.
- Ici, `append()` et `extend()` sont appelées sur la liste numbers, pour ajouter un ou plusieurs éléments.
- Les types de variables vont proposer des méthodes différentes.

```
from typing import List

# Déclaration d'une liste, avec des valeurs initiales
numbers: List[float] = [10.5, 32.0, -2, 23/2]

# Déclaration d'une liste vide, puis construction au fur et à mesure
numbers: List[float] = []
numbers.append(10.5)
numbers.append(32.0)
numbers.extend([-2, 32/2])
```

2. Nouvelle structure de données : les listes

Manipulation – Parcours de liste

- Parcourir une liste revient à parcourir un objet itérable.
- La boucle `for` est donc appropriée ici.

```python
values: List[int] = [0, 2, 7, 10, 15, 15, 19, 28, 29, 32, 32, 32, 38, 38, 50]

for i in range(len(values)):
    print(values[i])

for value in values:
    print(value)
```

2. Nouvelle structure de données : les listes

Manipulation – traitement sur les éléments

- Écrire une fonction `find(l: List[int], i: int) -> int` qui recherche la première position de l'entier `i` dans la liste `l`, ou retourne -1 si cet élément n'est pas dans la liste

```
values: List[int] = [0, 2, 7, 10, 15, 15, 19, 28, 29, 32, 32, 32, 38, 38, 50]

print(find(values, 7))   # Affiche 2
print(find(values, 32))  # Affiche 8
print(find(values, 5))   # Affiche -1
```

2. Nouvelle structure de données : les listes

Manipulation – traitement sur les éléments

```python
def find1(l: List[int], i: int) -> int:
    for index in range(len(l)):
        if l[index] == i:
            return index
    return -1


def find2(l: List[int], i: int) -> int:
    for index, element in enumerate(l):
        if element == i:
            return index
    return -1


def find3(l: List[int], i: int) -> int:
    if i in l:                  # Si i est présent dans l
        return l.index(i)       # Retourner la valeur de l'indice de la première occurrence de i
    return -1
```

2. Nouvelle structure de données : les listes

Manipulation – compter les éléments

- Écrire une fonction `count(l: List[int], i: int) -> int` qui compte le nombre d'occurrences de l'entier `i` dans la liste `l`

```
values: List[int] = [0, 2, 7, 10, 15, 15, 19, 28, 29, 32, 32, 32, 38, 38, 50]

print(count(values, 7))   # Affiche 1
print(count(values, 32))  # Affiche 3
print(count(values, 5))   # Affiche 0
```

2. Nouvelle structure de données : les listes

Manipulation – compter les éléments

```
def count(l: List[int], i: int) -> int:
    occurrences: int = 0
    for element in l:
        if element == i:
            occurrences += 1
    return occurrences


values: List[int] = [0, 2, 7, 10, 15, 15, 19, 28, 29, 32, 32, 32, 38, 38, 50]

print(count(values, 7))   # Affiche 1
print(count(values, 32))  # Affiche 3
print(count(values, 5))   # Affiche 0
print(values.count(7))
print(values.count(32))
print(values.count(5))
```

2. Nouvelle structure de données : les listes

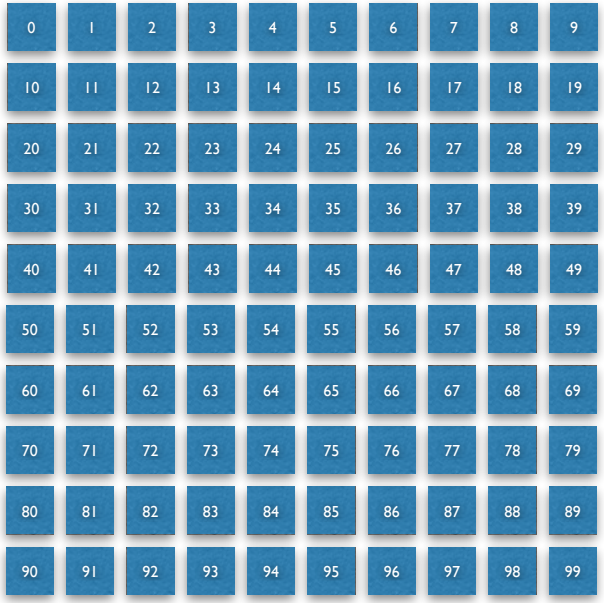
Quelques méthodes utiles

```
my_list.append(x)           # Ajoute l'élément x en fin de liste
my_list.extend([x, y, z])   # Ajoute les éléments x, y, z en fin de liste
my_list.insert(index, x)    # Insère l'élément x à l'indice index

my_list.remove(x)           # Supprime la première occurrence de x
my_list.clear()             # Supprime toute la liste

my_list.count(x)            # Compte le nombre d'occurrences de x
my_list.index(x)            # Retourne l'indice de la première occurrence de x, ou une erreur
my_list.sort()              # Trie la liste

my_list[:3]                 # Récupère la sous-liste d'éléments d'indice 0, 1, et 2
my_list[5:]                 # Récupère la sous-liste d'éléments d'indice 5, ..., n
my_list[3:6]                # Récupère la sous-liste d'éléments d'indice 3, 4, 5
```

2. Nouvelle structure de données : les listes

Application «concrète» : Problème des 100 prisonniers

- 100 prisonniers sont dans des cellules numérotées de 0 à 99
- Les clés de cellules sont mélangées et placées aléatoirement dans des boîtes, elles aussi numérotées de 0 à 99.
- Chaque prisonnier ne peut ouvrir que 50 boîtes
- Si tous les prisonniers trouvent leurs clés, alors ils peuvent s'échapper. Si un seul ne trouve pas, alors ils restent enfermés.
- Approche naïve : $\frac{1}{2^{100}}$
- Approche optimale : environ 31% !
- https://www.youtube.com/watch?v=iSNsgj1OCLA



- Ce problème sera modélisé et simulé la semaine prochaine !